IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE:

APPLICATION OF	:	ROBERT M. MORRIS & LEET E. DENTON, III
TITLE	:	A Visually Oriented Computer Implemented Application Development System Utilizing Standardized Objects And Multiple Views
APPLICATION NO.	:	09/184,738
FILING DATE	:	November 2, 1998
ART UNIT	:	2122
EXAMINER	:	Hoang-Vu Antony Nguyen-Ba
ATTORNEY DOCKET NO.	:	3042-3

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA

DECLARATION UNDER 37 C.F.R. 1.132

Michael L. Brachman, Ph.D. declares as follows:

1. A brief outline of my educational background and accomplishments is set out below:

Education:

B.S. in Psychology, Univ. of Michigan, 1974
Ph.D. in Sensory Science, Minor in Computer Science, Syracuse Univ., 1980
Post-doctoral Fellow, Northwestern Univ., 1981 (one year)

Present Employment:

President, Vega Applications Development, Inc. (since March 2000)

Previous Employment

Vice President, Information Technology, Managed Healthcare Associates Inc (March 1999 – March 2000)

President, Visual Software Solutions, Inc. (March 1995 – March 1999)

Partner and Director of Development, Micro Endeavors, Inc. (September 1989 – January 1995)

Books, Edited and Authored

Creating FoxPro Applications, Que Publishing Corporation, 1993
The FoxPro 2.5 API, Pinnacle Publishing, Inc., 1994

Refereed Journal Publications and Refereed Conference Publications:

A listing of these publications is attached as an appendix to this Affidavit.

2. I have reviewed the above referenced patent disclosure including the software source code appended to the disclosure and the claims.

3. I have been provided with a copy of the patent office action dated January 5, 2004 in which the Examiner stated that:

> Paragraph 9: "...certain claims of the invention are not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention."
>
> Paragraph 10: "The claim(s) contains subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with which it is most nearly connected, to make and/or sue the invention. Specifically, Applicants disclose in the specification at page 8, lines 1-3 that the manner in which their invention wraps the objects is equally usable with object written to standards other than the Microsoft VBX control and at page 15, line 16 to page 16, line 8 that the system of their invention wraps all objects in a kind of "envelope" of special properties that regulate how the objects act in the system. However, Applicants fail to disclose how to implement the wrapping in such a way to enable one skilled in the art to make and use the invention.
>
> The same reasoning also applies to the limitation means *for utilizing the additional properties and events to link and sequence the objects* claimed in these claims."

4. I have been asked to examine the patent documents, in particular the software source code, to determine whether the features referenced by the examiner are reasonably conveyed by the source code to a person skilled in the art. I was provided with a copy of the source code used to embody the invention.

5. I will address my comments with reference to each paragraph and subsection of the examiner's comments.

6. Examiner's paragraph 9.a:

After having examined the source code thoroughly, it is readily apparent that anyone skilled in the art will understand how objects and scripts are maintained separately from each other and from the executable files. This is a distinct advantage and the invention works perfectly well keeping these elements synchronized but separate.

The scripts themselves are maintained on an external storage device in a script file. The script file is separate from the objects that the script references. The script file is also separate from the executable files that utilize the script files. The files DOC_SCRI.H and DOC_SCRI.CPP use a symmetrical method of serializing the script both on input and output. The header of the file is organized by the file DOC_HEAD.CPP and describes the VBX controls invoked by the script. Of particular note is the CDocHeader::SetVBXFiles method, line 189. It is clear to me and to someone skilled in the art that saving the names of the VBX components does not constitute saving the VBX components themselves. The letters VBX in the name of the method in no way constrain this method to just VBX objects. It is clear to me that components other than VBX-type components would also work perfectly well with this method.

With the VBX objects, the file DOC_SCRI.CPP (line 224) calls a routine named CVBInstance::Serialize contained in the file VB_SERIA.CPP for each instance of a control in the script. This method shows clearly how to store the values of the properties of a VBX control in a script file. It is also clear that this method could be used to store the values of the properties for any type of control, not just VBX controls. When the program reads in a script file, a method in DOC_SCRI.CPP called CDocScript::LoadNeededVBXFiles() loads in the VBX controls from their separate files. This method can be utilized to load in any control of any type when required by the script. It is clear from the code that each control is expected to be stored separately prior to loading.

Accordingly, it is taught that not only are the scripts and controls kept separately from each other and from the executables, but also the methods to do this can easily be utilized for other objects by someone skilled in the art.

7. Examiner's paragraph 9.b:

The term "wrap" means that an additional layer is imposed upon an object and the user or programmer's access to that object. It is easy for anyone skilled in the art to understand from the source code how objects are wrapped. Such a wrapper allows the user/programmer to store and retrieve values for intrinsic properties of the object as well as add additional properties and events. There is no upper limit to the number of properties or events one could add using this method.

Wrapping is accomplished by the file VB_EXTRA.H that contains the definitions for the extra properties. The specific source code I examined lists a specific number of additional properties beyond those that are native to the VBX control. It would be trivial to increase or decrease this number by any amount. The extra property names are kept in a data structure along

with additional information that indicates the type of the additional property as well as flags to control its behavior. To add additional properties over and above those supplied, one would simply alter the NUMEXTRAPROPS define on line 74 of file VB_EXTRA.H. For each new property added, a corresponding addition would have to be made in the section for EXTRAPROPS. Similarly the extra properties have an enumerated type in line 32 of the same file, which would need to be changed as well. As mentioned above, this would be a trivial exercise for even a middle-level experienced programmer.

Events can be added in the following way: the class CMyVBXControl is derived from CVBControl in VB_INSTA.H. CMyVBXControl is a container for a CVBControl which is the Microsoft VBX control class. The additional events are added on lines 76-79 of VB_INSTA.H. This code declares handlers for the new events. If one wanted to add new handlers, it is obvious that new handlers could be declared here. The implementation of the handlers is in VB_INSTA.CPP in the lines following line 1031. Any additional handlers would be defined here. In this way, events can originate or be requested from the VBX by sending them a message. The CMyVBXControl will reflect the events to its parent, making the event seem to come from the control itself.

Event sequences can be triggered by a variety of methods. Once an event sequence is triggered, it is handled by the Sequence Execution Manager or SEM. Some events such as those that wait for button clicks or key presses are implemented by the Sequence Execution Manager and their implementations may be found in EV_SEM.H and EV_SEM.CPP. The events a control can fire are listed in the m_csaEventNames CStringArray member of CVBXRegInfo class, which is a central repository of information on object capabilities. The repository itself is defined and populated in VB_REGIS.CPP.

Accordingly, it is clearly taught to one skilled in the art how objects may be wrapped with additional properties and events beyond those internal to the object.

8. Examiner's paragraph 9.c:

Someone skilled in the art will readily understand from the source code how to utilize the additional events to link and sequence the objects by the following.

As mentioned previously, once a series of events is triggered, subsequent events are managed by the Sequence Event Manager. The code that teaches these capabilities is located in the files EV_SEM.H, EV_SEM.CPP, EV_TESC.H and EV_TESC.CPP. The EV_SEM files declare and implement the Sequence Execution Manager. In particular CSEMInstance::WaitForInput demonstrates how the program waits for input before proceeding. Also, CSEMInstance::WaitForDestroy waits for an event to destroy the VBX control's window before proceeding, etc. A key structure in VB_INSTA.H is m_cwaNextLine that is indexed by event. This is where the Branch-On-Realize figures out where to go next. Branch-On-Realize triggers whenever an object is instantiated. Instructing an object to pass execution to another object is by definition the link between the two.

Accordingly, it is clearly taught to one skilled in the art how to utilize additional events to link and sequence the objects.

9. Examiner's paragraph 9.d:

The essence of this system is message-based and the messages themselves contain information describing an event as it pertains to that object. Therefore, someone skilled in the art will readily understand from the source code how an event generated by an object triggers an instance of another object by the following.

The code clearly teaches how an event generated by an object can cause an instance of another object to be created via the appropriate message. Another object is instantiated when a VBX message is passed up to the output window. The COutputView::OnOutputVBXMessage handler schedules the next control instance on line 1147 of OUTPUTVI.CPP. The source code has the appropriate control measures so that if an object is only to be instantiated once this is indeed what happens.

Accordingly, it is clearly taught to one skilled in the art how an event generated by an object triggers an instance of another object. Since all communications between objects and by extension, by event sequences, are communicated by messages, there is no upper limit to the effect that such messages have. Creating an instance of an object is merely a simple example of the effect a message can have.

10. Examiner's paragraph 9.e:

One of the features of the system is parallel processing. Someone skilled in the art will readily understand from the source code how the system of the invention allows parallel processing.

Message-based systems such as the Windows operating system manage messages in a queue and dispatch the messages to the appropriate "listener" in order. Once an object receives a message, it begins processing that message. While this is happening, the next message in the queue is dispatched to the next object and that object or control begins processing as well. If there is a single processor, then slices of processor time are shared (weighted by thread priority) to all objects or processes currently executing instructions. Therefore, to the extent that any single processor system is capable of parallel processing, this system exploits that fact and also implements parallel processing. In a multi-processor machine, true parallel processing occurs because each processor can handle its own set of threads while the other processors are executing theirs. Within the proposed system, events trigger messages to other objects. A single event can trigger multiple messages to multiple objects. The Timed Event Scheduler can also trigger events. To the extent physically possible by the hardware, all of these event sequences can occur in parallel. Therefore, it is obvious to me that the underlying structure and design of the system inherently permits parallel processing. Each of the objects (controls) communicates through a private and predefined interface (properties and events). This allows the controls to freely execute without hindrance from one another. The message-passing schema coordinates the various processes.

Accordingly, it is clearly taught to one skilled in the art that a system in which each of the objects communicates independently through a predefined interface and executes without hindrance from other objects inherently implements parallel processing.

11. Examiner's paragraph 9.f:

To myself and someone skilled in the art will, the source code clearly demonstrates how development playback review is achieved in the system of the invention by the following.

Playback review is accomplished by the OnUpdate handlers of the various view classes. Each view supplied has its own OnUpate handler. The source code teaches how multiple views can be synced together. This happens when the Sync method of the appropriate view is called. The Sync method sets a flag, which is used by the OnUpdate method. For instance, in the icon view, also known as the MapperView, SetCurrentNode is called to highlight the current control when sequence execution reaches that point. The OnUpdate handlers of the other views proceed similarly. This method can be extended to cause any number of actions to occur to enhance the Playback review process.

Accordingly, it is clearly taught to one skilled in the art that development playback is implemented through OnUpdate handlers for each view utilized.

12. Examiner's paragraph 9.g:

To myself and someone skilled in the art, it is readily understandable from the source code that all different kinds of objects may be integrated into the system including those objects that implement program constructs and sub-languages. The objects could encapsulate complete processes written in any language and treated as a "black box" or the underlying language could be exposed for manipulation or alteration.

Sub-languages utilizing objects could more succinctly be described as languages implemented in objects such as VBX controls. Since the capabilities any object-oriented system stem from the capabilities of the objects, and there are no particular restrictions placed by the system on the function of those objects, it is clear to anyone skilled in the art that a sub-language can be implemented in a control and added to the system. Similarly, a program construct can be implemented in a control and added to the system. As a trivial example consider an "if/then" control. When the control is realized: if property A equals property B, an event is fired. A more elaborate object could expose an entire algorithm with the parameters exposed or even allow manipulation of the algorithm itself. The only requirement is that the properties to be altered must be exposed.

Accordingly, it should be clear from these examples that the software teaches how extensive sub-languages or objects written in alternate languages can be incorporated by those skilled in the art.

13. Examiner's paragraph 9.h:

It was readily understandable to myself and to someone skilled in the art, by examining the source code, one can see that the icon for each object is highlighted as the object is instantiated by the following.

Within the program MAPPERVI.CPP, at line 528, the procedure called CMapperView::SetCurrentNode is called. This in turn calls DrawIconSelection on line 423 for the same program. This highlights the icon. The original icon was draw by the code found on

line 226 by CMapNode::DrawIcon which is in MAPNODE.CPP. The bitmaps for the icons come out of the VBX controls themselves. The exact method for extracting an icon from an object was documented by Microsoft in documents that were readily available at the time the application was filed.

Accordingly, it should be clear that the software teaches one skilled in the art how to highlight an object's icon when the object is instantiated.

Based on the above references, I conclude that the source code disclosed to a person reasonably skilled in the art that the inventors were in possession of the methods set forth in the claims.

14. Examiner's paragraph 10:

Someone skilled in the art will readily understand from the source code how to make and/or use the invention. I actually did this. As noted above in response to examiner's paragraph 9.b, the source code fully discloses how to wrap controls with additional properties and events and how to use those controls in the system. As noted above in response to examiner's paragraph 9.c, the source code fully discloses how to utilize the additional properties and events to link and sequence objects. All comments with respect to the various subsections of examiner's paragraph 9 are incorporated and repeated here.

Provision of the source code for the invention constitutes a disclosure sufficient to use the method taught. To one skilled in the art, with the appropriate compiler and libraries installed (i.e. the Microsoft C++ development tools) of the time, the supplied materials would be sufficient to construct and use the invention. In order to prove to myself that the source code enabled the features I have attested to and performed as I understood it, I compiled a running version of the program which embodied and demonstrated the features I stated above.

I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statement may jeopardize the validity of the above referenced application or any patent issuing thereon.

Date: 7/6/04

Michael L. Brachman, Ph.D.

NOTARIAL SEAL
GEORGE D. MADONNA, Notary Public
Upper Providence Twp., Delaware County
My Commission Expires April 18, 2005

Bibliography by Michael L. Brachman, Ph.D.

Chemistry

Brachman, M. L. October 1972. Ion, My Love, Chemistry. 45: 32

Limulus

Barlow, Bolanowski and Brachman Science

Audition

Smith and Brachman 1977 JASA 62

Brachman, M. L. and Smith, R. L. 1979. Dynamic Versus Static Characteristics of Single Auditory Nerve Fibers. Soc. Neurosci. Abstr. 5: 16(A).

Barlow, Chamberlain, Brachman Science 1979

Smith and Brachman 1980 HR 2

Smith and Brachman 1980 BR 184

Smith and Brachman 1980 Van den Brink

Brachman 1980 Dissertation

Brachman, M. L, Smith, R. L. and Frisina, R. D. 1979. Effects of Time on Decremental Responses of Single Auditory Nerve Fibers, J. Acoust. Soc. Am. 65: S83(A).

Smith and Brachman Cyber 1982

Smith, R. L., Brachman, M. L. and Goodman, D.A. 1983. Adaptation in the Auditory Periphery, Annals New York Academy of Science. 79: 79-93.

FoxPro

Brachman, M. L. March 1993. Overcome FoxPro's Screen Painter Limitations, FoxPro Advisor, 1: 28.

Brachman, M. L. April 1993. Extending FoxPro, Dr. Dobb's Journal, pp. 35 - 38.

Brachman, M. L. May 1993. Using FoxPro's API: Adapting Existing DOS Programs, Data Based Advisor, 11: 162-167.

Goley, G. F. and Brachman, M. L. 1993. Creating FoxPro Applications, Que. Carmel, IN.

Brachman, M. L. 1994. The FoxPro 2.5 API, Pinnacle Publishing, Inc. Kent, WA.

Brachman, M. L. July 1995. Use Column Registration Functions with BROWSE, FoxTalk, pp. 23 - 26

Brachman, M. L. and Willhoite, J. August 1995. The Incredible, Expandable BROWSE, FoxTalk, pp. 10 - 13.